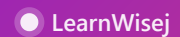


Theming, Ressourcen, Lokalisierung und UI-Modernisierung

Vertiefung des Themings: Wechsel zwischen hellem und dunklem Theme zur Laufzeit, wiederverwendbare Status-Chips über CSSStyle, Verwaltung von Ressourcen und kulturabhängige Lokalisierung.



Praxisziel

Modernisieren Sie das Operations-Dashboard von TicketOps von Anfang bis Ende: Wechseln Sie zur Laufzeit zwischen hellem und dunklem Theme, bauen Sie ein wiederverwendbares Status-Chip-Control, das über CSSStyle gestaltet wird, verwalten Sie Icons und Bilder als Ressourcen, und lokalisieren Sie Beschriftungen und Formatierung für zwei Kulturen.

1. Themes zur Laufzeit mit Application.Theme wechseln

Die TicketOps Console bietet im Header einen Umschalter, mit dem Sachbearbeiter zwischen einem hellen Theme für Tagschichten und einem dunklen Theme für die nächtliche Leitstelle wechseln können. Weisen Sie `Application.Theme` einen Theme-Namen zu, und Wisej.NET rendert jedes offene Fenster neu; Controls müssen Sie nicht selbst neu aufbauen.

Halten Sie die Theme-Namen an einer Stelle, und speichern Sie die Wahl des Sachbearbeiters, damit das Dashboard nach einer Aktualisierung der Session im selben Theme wieder öffnet.

```
// TicketOps-Header: ein einziger Umschalter steuert die ganze Konsole
private void themeSwitch_CheckedChanged(object sender, EventArgs e)
{
    // Integrierte Wisej.NET-Theme-Namen, die mit der App ausgeliefert werden
    var theme = themeSwitch.Checked ? "BootstrapDark-4" : "Bootstrap-4";

    Application.Theme = theme; // gestaltet alle offenen Fenster live neu
    Application.Session["ticketops.theme"] = theme; // pro Session merken
}

// Beim Start das letzte Theme wiederherstellen, bevor das Dashboard angezeigt wird
protected override void OnLoad(EventArgs e)
{
    if (Application.Session["ticketops.theme"] is string saved)
        Application.Theme = saved;

    base.OnLoad(e);
}
```



Gewohnheit für den Produktivbetrieb

Verstreuen Sie Theme-Namen nie als Literale über die Masken. Halten Sie `"Bootstrap-4"` und `"BootstrapDark-4"` in einer Konstanten, damit ein Rebranding eine Änderung in einer Zeile ist.

2. Ein wiederverwendbarer Status-Chip, gestaltet über CSSStyle statt fest codierter Farben

Jede Zeile eines Arbeitsauftrags zeigt einen Status: *Open, In Progress, Blocked, Resolved*. Statt Labels auf jeder Maske von Hand einzufärben, bauen Sie ein einziges `StatusChip`-UserControl. Es stellt eine Eigenschaft `Status` bereit und setzt ein CSSStyle-Klassen-Token; die eigentlichen Farben leben im Theme, sodass sowohl das helle als auch das dunkle Theme den Chip korrekt darstellen.

```
// StatusChip: ein wiederverwendbares UserControl für das Operations-Dashboard
public partial class StatusChip : UserControl
{
    private WorkOrderStatus _status;

    public WorkOrderStatus Status
    {
        get => _status;
        set
        {
            _status = value;
            labelText.Text = value.ToString();

            // Das Aussehen über ein CSSStyle-Token steuern, nicht über Inline-Farben
            // Das Theme definiert .chip-open, .chip-blocked usw. für jedes Theme
            this.CssClass = "chip chip-" + value.ToString().ToLowerInvariant();
        }
    }
}
```

Weil der Chip einen Klassennamen referenziert und das Theme die Palette liefert, sieht dasselbe Control bei Tag und bei Nacht richtig aus – ohne eine einzige Bedingung in Ihrem Code.

3. Entscheiden, wohin ein Stil gehört

Der häufigste Fehler bei der Modernisierung ist Farbe in der falschen Schicht. Entscheiden Sie anhand dieser Tabelle, wo jede visuelle Regel lebt, bevor Sie sie schreiben.

Technik	Gültigkeitsbereich	Verwenden für	Nicht verwenden für
Theme / Theme Builder	Ganze Anwendung	Palette, Typografie, Standard-Aussehen der Controls, Chip-Farben	Einmalige Anpassungen an einem einzelnen Control
CSSStyle / CssClass	Ein Control oder ein wiederverwendbares UserControl	Gemeinsame Komponentenvarianten wie den Status-Chip	Anwendungsweite Farbentscheidungen
Inline BackColor / ForeColor	Eine Control-Instanz	In modernen Masken fast nichts	Alles, was Sie später neu gestalten müssen



Typischer Fehler: fest codierte Farben

Ein Chip mit `chip.BackColor = Color.Red` bleibt auch im dunklen Theme rot und zerstört den Kontrast. Leiten Sie das Aussehen über das Theme oder eine CssClass, damit beide Themes korrekt bleiben.

4. Icons und Bilder als Ressourcen verwalten

Die Toolbar des Dashboards braucht ein Aktualisieren-Icon, ein Export-Icon und ein Status-Symbol pro Chip. Behandeln Sie diese als verwaltete Ressourcen statt als lose Dateipfade. Wisej.NET kann theme-abhängige Bilder auflösen, und die Wiederverwendung einer einzigen Ressourcenreferenz hält die Toolbar über alle Masken hinweg einheitlich.

Ressourcentyp	Am besten geeignet für	Wie er referenziert wird
Theme-Image-Mixin	Icons, die sich je nach Theme unterscheiden (helle vs. dunkle Symbole)	Über den Bildnamen im Theme, automatisch aufgelöst
Eingebettete Ressource / Projektressource	Logos und Bilder, die sich nie je nach Theme ändern	Über den generierten Ressourcen-Accessor
Externe Datei / URL	Vom Benutzer bereitgestellte Inhalte wie Anhänge	Zur Laufzeit über Pfad oder URL

```
// Toolbar-Buttons beziehen verwaltete Ressourcen, keine verstreuten Pfade
refreshButton.ImageSource = "resource.wx/TicketOps.Icons.refresh.svg";
exportButton.ImageSource = "resource.wx/TicketOps.Icons.export.svg";

// Eine überall wiederverwendete Referenz hält die Toolbar einheitlich
public static class Glyphs
{
    public const string Refresh = "resource.wx/TicketOps.Icons.refresh.svg";
    public const string Export = "resource.wx/TicketOps.Icons.export.svg";
}
```

5. Beschriftungen und Formatierung für zwei Kulturen lokalisieren

TicketOps läuft für englische und italienische Operations-Teams. Zwei Dinge müssen sich gemeinsam ändern: der **Text** der Beschriftungen (über lokalisierbare Ressourcen) und die **Formatierung** von Datumswerten und Währungen (über `CultureInfo`). Wer nur eines davon lokalisiert, erhält englische Wörter neben italienisch formatierten Zahlen – und das wirkt kaputt.

Markieren Sie Texte von Formularen und Controls als lokalisierbar, damit übersetzte `.resx`-Dateien jede Kultur beliefern. Formatieren Sie Werte ausdrücklich mit der Kultur des Sachbearbeiters, statt sich auf die Standardeinstellung des Servers zu verlassen.

```
// Kulturabhängige Formatierung für das Detail-Panel eines Arbeitsauftrags
using System.Globalization;

// en-US → 7/3/2026, $1,250.00   it-IT → 03/07/2026, 1.250,00 €
var culture = new CultureInfo(operatorCultureCode); // "en-US" oder "it-IT"

createdLabel.Text = workOrder.CreatedOn.ToString("d", culture);
costLabel.Text    = workOrder.LaborCost.ToString("C", culture);
slaLabel.Text     = workOrder.DueBy.ToString("g", culture);

// Die Kultur des Requests setzen, damit auch lokalisierbare Ressourcen aufgelöst werden
Application.CurrentCulture = culture;
Application.CurrentUICulture = culture;
```



Gewohnheit für den Produktivbetrieb

Setzen Sie `CurrentUICulture` für Texte und `CurrentCulture` für die Formatierung immer gemeinsam, und testen Sie dann die breiteste Beschriftung erneut. Italienische Texte sind oft länger als englische und können einen Chip oder Button mit fester Breite abschneiden.

6. Eine Checkliste zur UI-Modernisierung, angewendet auf das Dashboard

Prüfen Sie jede Operations-Maske anhand dieser Fragen, bevor Sie sie modern nennen. Sie übersetzen die abstrakten Regeln aus dem Lektionsleitfaden in konkrete Prüfungen an echten TicketOps-Masken.

- ✓ Stammt jede Statusfarbe aus dem Theme oder einer CssClass, ohne Inline- `BackColor` ?
- ✓ Gestaltet der Hell/Dunkel-Umschalter die ganze Konsole neu, einschließlich Chips und Toolbar-Icons?
- ✓ Werden alle Toolbar-Icons aus verwalteten Ressourcen statt aus losen Pfaden aufgelöst?
- ✓ Ist jede sichtbare Beschriftung lokalisierbar, und formatiert das Panel Datumswerte und Währungen über `CultureInfo` ?
- ✓ Passt die breiteste übersetzte Beschriftung in beiden Themes noch, ohne abgeschnitten zu werden?
- ✓ Könnte ein künftiges Rebranding die Palette ändern, ohne einzelne Masken zu bearbeiten?

Wichtige Begriffe

- ✓ **Application.Theme** – einen Theme-Namen zuweisen, um alle offenen Fenster zur Laufzeit neu zu rendern.
- ✓ **CssClass / CSSStyle** – ein Klassen-Token anhängen, damit ein wiederverwendbares Control die Farben des Themes übernimmt.
- ✓ **Status-Chip** – ein UserControl, das den Status eines Arbeitsauftrags über eine Klasse statt über Inline-Farbe ausdrückt.
- ✓ **Verwaltete Ressource** – ein Icon oder Bild, das über seinen Namen referenziert wird, damit es einheitlich und theme-abhängig bleibt.
- ✓ **Eigenschaft Localizable** – Control-Text, der aus kulturspezifischen `.resx`-Ressourcen stammt.
- ✓ **CultureInfo** – steuert die kulturabhängige Formatierung von Datum und Währung für jedes Operations-Team.